CASPER: An Integrated Energy-Driven Approach for Task Graph Scheduling on Distributed Embedded Systems

Vida Kianzad, Shuvra S. Bhattacharyya and Gang Qu
ECE Department and Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742
{vida, ssb, gangqu}@eng.umd.edu

Abstract

For multiprocessor embedded systems, the dynamic voltage scaling (DVS) technique can be applied to scheduled applications for energy reduction. DVS utilizes slack in the schedule to slow down processes and save energy. Therefore, it is generally believed that the maximal energy saving is achieved on a schedule with the minimum makespan (maximal slack). Most current approaches treat task assignment, scheduling, and DVS separately. In this paper, we present a framework called CASPER (Combined Assignment, Scheduling, and PowER-management) that challenges this common belief by integrating task scheduling and DVS under a single iterative optimization loop via genetic algorithm. We have conducted extensive experiments to validate the energy efficiency of CASPER. For homogeneous multiprocessor systems (in which all processors are of the same type), we consider a recently proposed slack distribution algorithm (PDP-SPM) [3]: applying PDP-SPM on the schedule with the minimal makespan gives an average of 53.8% energy saving; CASPER finds schedules with slightly larger makespan but a 57.3% energy saving, a 7.8% improvement. For heterogeneous systems, we consider the power variation DVS (PV-DVS) algorithm [13], CASPER improves its energy efficiency by 8.2%. Finally, our results also show that the proposed single loop CASPER framework saves 23.3% more energy over GMA+EE-GLSA [12], the only other known integrated approach with a nested loop that combines scheduling and power management in the inner loop but leaves assignment in the outer loop.

1. Introduction

Energy consumption has become a major design issue for modern real-time embedded systems, especially battery-operated portable devices. Such systems also operate under tight and hard deadlines. While the early task completion (before the deadline) may not bring the system extra benefit, one can utilize the extra available time to improve other valuable system performance parameters such as energy consumption. Energy consumption is a quadratic function of the supply voltage and processor speed, and reducing the supply voltage and thus processing speed can save energy, but at the cost of increased execution time. Dynamic voltage scaling (DVS) is a promising method for embedded systems to exploit multiple supply voltage and clock frequency levels and to achieve the highest possible energy efficiency for time-varying computational loads while meeting the deadline constraint.

A large number of papers devoted to the energy-aware voltage-scheduling problem only consider single-processor systems or independent tasks (e.g., see [7]). There are also some research works that address dependent tasks on multiprocessors. However, in most of these works the authors propose that their algorithms are to be used in the inner loop of a system-level optimization tool and hence proceed with the assumption that either the whole process of task assignment to the processors and the ordering of tasks or one of these steps (task assignment or task ordering) is determined a priori and do not factor the effect of ordering or assignment in the DVS results [6][9][11]. One serious drawback to this assumption is that globally optimal voltage scheduling may not be generated. We believe that the integration of task assignment and ordering and voltage scheduling is essential since different assignments and orderings provide voltage schedulers with great flexibility and potential energy saving that can be achieved. In this paper, we address such integration that simultaneously optimizes for task assignment, ordering and scheduling.

The idea of integrating scheduling into the power management process has been studied for heterogeneous multiprocessor systems. The work in [13] employs two nested genetic algorithms (GAs) where the outer GA generates the assignments and the inner one explores various orderings. This algorithm is not however efficient in terms of run time. Furthermore, little research has been done for such integration for the homogeneous multiprocessor case. Although the homogeneous scenario can typically be handled as a special case of techniques that address heterogeneous multiprocessor systems, one can expect that when we limit the target architecture to homogeneous processors, the result would better reflect the effect of ordering on DVS as the effect of processor selection and assignment has been toned down. Additionally, all the available compile time is spent on optimizing the task ordering and scheduling rather than being shared with allocation and assignment. In our approach, we present a GA framework that can be applied to both heterogeneous and homogeneous multiprocessor systems. It thoroughly searches the solution space to find an assignment and ordering of tasks on each processing element (PE) and generates a schedule that meets the deadline and minimizes the power consumption simultaneously. We study the impact of i) integrating the scheduling process into the power optimization framework for DVS-enabled embedded multiprocessor systems and ii) combining task mapping, ordering and scheduling and encoding them in the form of a single chromosome in a GA framework.

2. Problem Statement and Assumptions

We consider an embedded application represented in terms of an acyclic task graph $G = (V, E)$ with $|V|$ tasks $\{v_1, v_2, ..., v_{|V|}\}$ and $|E|$ communication edges. The multiple processor system being used to implement the application consists of n_P processing elements (PE) of the following types: general-purpose processors, application-specific integrated circuits, and FPGAs. A communication resource (CR) is a hardware resource for communication messages. In the case of homogeneous systems, we assume all the PEs are general-purpose processors with identical CRs (links).

For the application, we further assume that there is a partial order among the tasks (vertices) indicating the data dependency. For example, $v_i < v_j$ implies that v_i has higher scheduling priority than v_j, which means that v_j can not start until v_i finishes. Tasks can have hard or soft deadlines. A hard deadline must be met at runtime to ensure the correctness and feasibility of the solution. We represent the set of tasks with hard deadlines as V_d. In addition, we define the following functions:

- $wcet : V \times PE \rightarrow \Re^+$ denotes a function that assigns the worst case execution time $(wcet(v_i, pe_j))$ to task v_i of set V running on processing element pe_j. In homogeneous multiprocessor systems this function is reduced to a one-dimensional function $wcet : V \rightarrow \Re^+$ (i.e. $wcet(v_i)$). The execution of tasks is assumed to be non-preemptive.

- $C : V \times V \times CR \rightarrow \Re^+$ denotes a function that gives the latency incurred on each communication edge on a given communication resource (CR). That is $C(v_i, v_j, cr_k)$ is the cost of transferring data between v_i and v_2 on communication resource cr_k if they are assigned to different processing elements. This value is zero if both tasks are running on the same processing element. $C(v_i, v_j, cr_k)$ is reduced to $C(v_i, v_j)$ when we consider a homogeneous communication network.

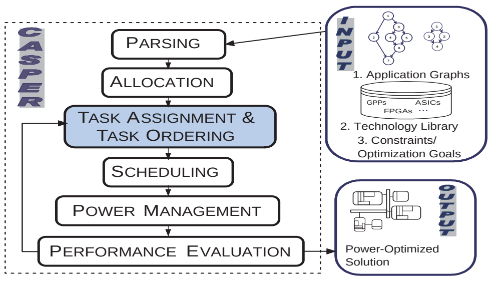


Figure 1. CASPER framework.

We adopt the following models for the DVS-enabled processor's dynamic power consumption p_d and operational frequency f:

$$\begin{aligned} p_d &= C_{ef} \cdot V_{dd}^2 \cdot f, & (1) \\ f &= k \cdot (V_{dd} - V_t)^2 / V_{dd}, & (2) \end{aligned}$$

where C_{ef} is the effective switching capacitance, V_{dd} is the supply voltage, k is a circuit dependent constant and V_t is the threshold voltage. The problem formulation remains the same and our approach are still applicable for other models. We use them mainly for the purpose of illustrating the idea and comparison with existing results where these models are used [3][12][13].

Given the application and multiprocessor system described as above, we find (i) a mapping of tasks to PEs, (ii) an ordering of the tasks and edges, and (iii) the voltage profile for each task such that all the hard deadline constraints are met and the total energy consumption is minimized.

3. Proposed Algorithmic Solution

Our proposed solution is an iterative improvement framework that integrates task assignment, ordering and scheduling, and static power management all in one phase. We call this framework Combined Assignment, Scheduling, and PowER-management or CASPER. A high-level overview of CASPER is depicted in Fig. 1.

CASPER takes the application task graphs, the sets of PEs and CRs, and the constraint/optimization requirements as input. It first parses these inputs and creates the appropriate data structures. Next, it uses a simple standard algorithm to allocate PEs and CRs such that every task and every edge has at least one instance of PE or CR that it can execute on. It then uses a genetic algorithm that combines the task assignment, ordering and scheduling, as well as power management by DVS to find the most energy efficient solution (see the loop in Fig. 1). Details on the genetic algorithm is given below in Section 3.1. Section 3.2 briefly describes the two power management techniques that we have selected for homogeneous and heterogeneous multiple processor systems. We mention that any slack distribution based power management method can be integrated into the CASPER framework. We selected these two because they provide the best available results.

3.1. Combined Assignment and Scheduling

The core part of CASPER is a genetic-list scheduling algorithm that encodes both assignment and ordering in a single chromosome (similar representation has been used for a multiprocessor scheduling algorithm called CGL [4]). In this representation, each individual solution or chromosome is encoded as a list of n_P strings, with each string corresponding to one allocated PE of the target system. The strings maintain both the assignment and execution order of the tasks on each PE. Fig. 2 illustrates the relationship between an arbitrary schedule for a task graph and its corresponding string representation for a homogeneous multiprocessor system. The list of tasks within each PEs of the schedule is ordered in ascending order of the task heights, which guarantees that the precedence constraints are satisfied. The height $height(v_j)$ of a task v_j is a random integer whose value is such that: $\forall v_i \in R_{v_j} \wedge \forall v_k \in U_{v_j}$:

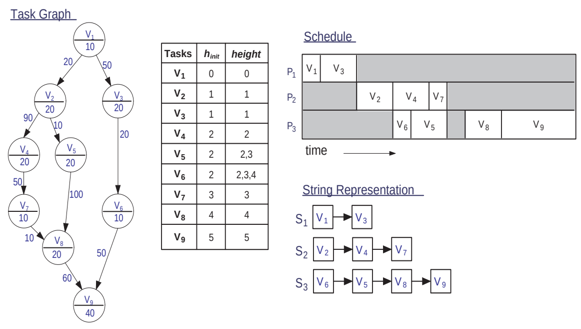
$$max(h_{init}(v_i)) + 1 \leq height(v_j) \leq \\ min(h_{init}(v_k)) - 1, \qquad (3)$$

where R_{v_j} (U_{v_j}) is the set of immediate predecessors (successors) of v_j and h_{init} is defined as,

$$h_{init}(v_i) = \begin{cases} 0, & R_{v_i} = \emptyset, \\ 1 + \underset{v_j \in R_{v_i}}{max}\, h_{init}(v_j), & otherwise. \end{cases} \qquad (4)$$

A randomized version of list scheduling is used to generate the initial population as follows: For each height $\hbar$ perform the following steps: (1) Pick a random task v_r from $V(\hbar)$ ($V(\hbar)$ is defined as the set of tasks in G with height $\hbar$). (2) Pick a processing element pe_r that can execute v_r at random. (3) Assign v_r to pe_r. (4) Repeat Steps (1)-(3) until all remaining tasks from $V(\hbar)$ are scheduled [4].

Once the population is generated, the chromosomes' fitness needs to be evaluated. The chromosome's fitness or performance measure consists of two parts: the first part is a measure of constraint satisfaction (satisfying the deadline) and the second part is based on the schedule performance with respect to energy E. This is because objective measures are, in practice, meaningless if the schedule



Tasks	h_{init}	height
V_1	0	0
V_2	1	1
V_3	1	1
V_4	2	2
V_5	2	2,3
V_6	2	2,3,4
V_7	3	3
V_8	4	4
V_9	5	5

$$\begin{tikzcd} S_1 & V_1 \arrow[r] & V_3 \end{tikzcd}$$

$$\begin{tikzcd} S_2 & V_2 \arrow[r] & V_4 \arrow[r] & V_7 \end{tikzcd}$$

$$\begin{tikzcd} S_3 & V_6 \arrow[r] & V_5 \arrow[r] & V_8 \arrow[r] & V_9 \end{tikzcd}$$

Figure 2. Illustration of the string representation of a schedule.

is infeasible (i.e., violates the constraints). Hence, the optimization measures should not be considered until the given constraint has been satisfied. The fitness value $F(I_i, P_t)$ for this problem with (time) constraint performance measure τ_c and (energy dissipation) optimization performance measure E_{opt} for each individual chromosome I_i in population P_t is defined in (5):

$$F_i(I_i, P_t) = \begin{cases} \frac{\tau_c(I_i,P_t)}{2} & \Delta_c(I_i, P_t) > 0, \\ \frac{1+E_{opt}(I_i,P_t)}{2} & \Delta_c(I_i, P_t) \leq 0, \end{cases} \qquad (5)$$

where

- $\tau_c(I_i, P_t)$ is the constraint performance measure and is defined as,

$$\tau_c(I_i, P_t) = \begin{cases} \frac{1}{1+\Delta_c(I_i,P_t)} & \Delta_c(I_i, P_t) > 0, \\ 1 & \Delta_c(I_i, P_t) \leq 0. \end{cases} \qquad (6)$$

  Here, $\Delta_c(I_i, P_t)$ is a measure of time constraint (deadline) violation and is defined as $\Delta_c(I_i, P_t) = \sum_{v \in V_d} (\tau_{end}(v) - \tau_d(v))$, where $\tau_{end}(v)$ is the finish time of task v in the schedule and $\tau_d(v)$ is task v's hard deadline.

- $E_{opt}(I_i, P_t)$ represents the fitness of the individual chromosome I_i with respect to the energy and is defined as

$$E_{opt}(I_i, P_t) = \frac{\max_i(E(I_i, P_t)) - E(I_i, P_t)}{\max_i(E(I_i, P_t))}. \qquad (7)$$

Most research works give the most weight to the solutions that have a larger global slack (the difference between the deadline and the makespan and do not consider local slack (gaps between the tasks) as important. Such techniques employ scheduling algorithms that find the minimum-length makespan and feed their results to the associated power management algorithms. However, we regard both global and local slack as equally important, and consequently use an integrated approach to find a solution that has an overall slack distribution (global and local) that saves the most energy. This can also be seen from Equation (5). The second condition of this equation shows that for all the solutions that satisfy the time constraint (or meet the deadline) the effect of constraint satisfaction is a constant number and not a function of the global slack.

It can also be observed from Equation (5) that infeasible solutions are allowed in the solution. Considering infeasible solutions in the intermediate states of optimization is to make the solution space as continuous as possible. In complex systems we expect that most of the obtained schedules are not feasible. If such schedules are not accepted as members of the population then we cannot guarantee that starting from any solution the entire solution space can be searched.

The selection process allows the algorithm to take biased decision favoring good solutions. We use the "roulette wheel" principle to randomly select an individual in population P_t. The better the fitness of the individual the better the odds of it being selected. The selected individuals are then crossed to make new solutions (a cutting place is decided based on a randomly chosen height). The mutation randomly transforms a solution to a new solution with a single exchange of two tasks in the scheduled solution. By use of the height values, crossover and mutation always maintain the precedence constrains and hence never generate any invalid solutions(see S6 and S7 in Fig. 3). Once the new individuals are generated, the genetic algorithm proceeds by evaluating the new solutions and repeating the same steps of selection, crossover and mutation until the termination condition is met (such as the maximum number of generation is reached or the energy saving in two consecutive generations is less than 1%).

The outline of our algorithm is presented in Fig. 3. The power management algorithms used in $S3$ are described in the following section.

```
INPUT: A task graph $G$, $n_P$ PEs and time-constraint $\tau_d$.
OUTPUT: An energy-optimized mapping of the task graph.
S1   Generate initial population $P_t$ (of size $POP\_SIZE$)
where each individual is a list of ordered strings of size $P$.
S2   Compute the finish times of tasks for each individual.
S3   Apply the power management algorithm to each individual and compute the corresponding energy dissipation $E$.
S4   Calculate each individual's fitness based on $\tau_c$ and $E_{opt}$.
S5   Select $k$ individuals from $P_t$ according to their fitness
values using a roulette wheel, where $k = POP\_SIZE$.
S6   Perform the crossover $\frac{k}{2}$ times: select 2 strings,
cut each string in 2 parts by randomly choosing a height $h$
and partitioning the tasks with heights larger and smaller than
$h$ into right and left sets respectively. Keep the left sets and
exchange the right sets to get two new strings.
S7   Perform the mutation randomly: choose task $v_i$, then
pick another task $v_j$ among all the tasks with the same height
as $v_i$ at random and exchange the position of the two tasks.
S8   If the maximum number of generations is reached stop,
otherwise go to $S2$.
```

Figure 3. Flow of CASPER

3.2. Power Management Techniques

In this part, we briefly introduce the power management algorithms that we use in our experimentation. Specifically, we use the *Static Power Management with Proportional Distribution and Parallelism Algorithm* (PDP-SPM) for homogeneous system and the Power Variation Dynamic Voltage Scheduling (PV-DVS) algorithm for heterogeneous systems. The only reason to use them (see S3 of Fig. 3) is that they have been reported to outperform other techniques in energy efficiency by a large margin. More details about these two algorithms can be found in [3] and [13] respectively. However, the proposed CASPER framework can adopt any existing power management methods.

3.2.1. Static Power Management with Proportional Distribution and Parallelism Algorithm (PDP-SPM).

PDP-SPM algorithm is a static power management (SPM) technique for homogeneous system to reduce energy consumption by utilizing slack, both global and local, and parallelism among the processors. For a scheduled task graph, it applies the following two phases repetitively: 1) proportionally distribute the slack among the tasks under the deadline constraint; and 2) create new (local) slack based on parallelism and return to the first phase to re-distribute it.

In the first phase, the algorithm distributes slack, both the global and local static slack, to the tasks hierarchically. First, the global slack is distributed to all vertices proportionally to their execution time. Each vertex will have its execution time scaled up by a factor of δ. However, this does not guarantee that the new makespan will be increased by the same factor δ because the inter-processor communication cost does not scale. Therefore, this process is applied repetitively until the new makespan violates the deadline τ_d. Then the CPU time assigned to all the vertices along critical paths will be scaled down to meet the deadline and marked as final. There may still exist local slack and hence the algorithm continues to scale up the execution time for those vertices that have not been marked as final. At the end of this phase, little or none slack is expected.

In the second phase, PDP-SPM re-allocates the CPU time assigned to each task based on the system's degree of parallelism (that is, the number of PEs running at the same time). The basic idea is to *create new slack* by reducing the CPU time assigned to the tasks with the minimal degree of parallelism. Such new slack will be redistributed using the same procedure as in the first phase. If this results in energy reduction, CPU time will be reduced from this same task again until little or no energy saving can be achieved. Then this process restarts with another task of the minimal degree of parallelism until all the tasks are examined.

3.2.2. Power Variation (PV) DVS Algorithm. For heterogeneous system, we consider PV-DVS algorithm, which reports significantly higher energy reduction than other DVS scheduling approaches [13]. This algorithm is based on a constructive heuristic using the energy difference ($\Delta E(v)$): the energy saving obtained by extending task v's execution time by a time quantum of Δt.

The algorithm first calculates the available slack times of each hard deadline task to identify all extendable tasks. Next, it calculates the slack time of all tasks and inserts all the tasks with a slack time greater than a Δt_{min} into a priority queue. The energy difference $\Delta E(v)$ for all the extendable tasks in the priority queue are then calculated and the queue is sorted in decreasing order of the energy differences (or tasks energy saving potential). The algorithm iterates until no extendable tasks are left in the priority queue.

In each iteration the algorithm picks the first element of the priority queue and extends it by Δt and updates the energy dissipation value of the selected task. The extension is then propagated through the mapped and scheduled task graph. Next, the inextensible tasks are removed from the extendable task priority queue. Taking into account the tasks in the priority queue the time quantum Δt is recalculated, energy differences are updated and priority queue is reordered. At this point, the algorithm either invokes a new iteration or ends, based on the state of the extendable queue.

4. Experimental Results

The goal of our experiments is twofold: (i) to measure the effectiveness of an integrated framework versus the one that separates task assignment, ordering, and power management; (ii) to evaluate our integrated framework CASPER against another synthesis approach [13], which is the current state-of-the-art.

For the first goal, we compare CASPER with the Heterogeneous/Homogeneous Genetic List Scheduling (HGLS). HGLS is the same as CASPER except that the power management phase is moved out from the optimization loop. Therefore, the genetic algorithm finds a solution that is optimized for makespan, on which the power management technique will be applied.

For the second goal, we mention that synthesis approach proposed in [13] separates task mapping (assignment) and scheduling into two nested optimization loops. The outer loop (GMA) is a genetic algorithm optimizing for mapping, and the inner loop (EE-GLSA) is an energy efficiency Genetic List Scheduling Algorithm. We hereby refer to this approach as GMA+EE-GLSA.

All algorithms were implemented using LEDA, a C++ class library of efficient graph-related data structures and algorithms, on an Ultra SPARC-IIi/440MHz. The GA parameters are set as follows: population size = 70 with 50% generation overlap, mutation rate = 0.2 and crossover rate = 0.8. We used different sets of benchmarks for homogeneous/heterogeneous target architectures as follows:

I) The homogeneous set consists of two subsets:

- The first set is the Referenced Graph (RG) set that includes task graphs that have been used by different researchers for a similar application. This set consists of 10 task graphs that are represented as RG1-RG10. RG1 and RG2 are taken from [1] and [2], respectively. RG3 is a quadrature mirror filter bank, RG4 is based on gaussian elimination for solving four equations in four variables [10], RG5 and RG6 are different implementations of the fast Fourier transform (FFT), RG7 is an adaptation of a PDG of a physics algorithm [10], RG8 is an implementation of the Laplace transform [15], RG9 is another implementation of FFT and RG10 is based on mean value analysis [8]. The deadline assigned to each graph was computed using a method similar to that used in [5] based on the graph's maximum length path and the average execution times of the tasks.

- The second set is the TG set and consists of 5 large random task graphs ($50 \sim 100$ nodes) that were generated using TGFF [5].

II) The heterogeneous set consists of 25 TGFF generated task graphs (tgff1 - tgff25) used by Schmitz et al. [13]. The specification includes graphs of 8 to 100 task nodes that are mapped to heterogeneous architectures containing *power managed* DVS-PEs and non-DVS enabled PEs. Accordingly, the power dissipation varies among the executed tasks (with maximal variation of 2.6 times on the same PEs).

4.1. Homogeneous System

To evaluate the effectiveness of the integration process, we first ran HGLS, for a given number of generations (500 generations here). Once HGLS generates the final solution (a schedule with minimum makespan), we apply the PDP-SPM algorithm to this result and measure the energy saving for the schedule. Next we run CASPER for the same number of generations, using the same PDP-SPM algorithm as the power management method in S3 (Fig. 3) and find a schedule that minimizes the energy consumption while meeting the deadline. We then compare the results. It should be noted that both algorithms indeed use the same task assignment and scheduling scheme with the difference that HGLS generates the minimum-makespan schedule with no regard to energy saving while CASPER finds a schedule that consumes less energy. Scheduling and power

Table 1. Energy saving by CASPER and HGLS for RG and TG set.

Task Graph	$\|V\|/\|E\|$	τ_d	HGLS + PDP-SPM μ	HGLS + PDP-SPM % saving	Proposed (CASPER) μ	Proposed (CASPER) % saving	Proposed (CASPER) %improv.
RG1	16/24	65	44	57.4	45	60.7	7.8
RG2	17/28	50	37	49.1	38	54.3	10.2
RG3	14/15	130	102	41.0	102	44.0	5.1
RG4	20/39	2120	1596	50.4	1597	52.3	3.7
RG5	28/32	225	150	57.4	151	61.5	9.5
RG6	28/32	460	265	64.1	265	65.5	4.1
RG7	41/69	925	585	58.5	610	62.2	9
RG8	18/29	665	390	62.0	420	65.6	9.3
RG9	95/158	151	118	47.1	122	50.5	6.4
RG10	361/684	17154	11933	58.8	12818	62.2	8.1
TG1	43/74	1400	1014	47.1	1025	50.5	6.5
TG2	68/119	2000	1345	57.1	1353	59.3	5.3
TG3	93/170	3300	2462	49.3	2472	53.5	8.4
TG4	93/170	3300	2132	59.5	2172	67.3	19.3
TG5	113/216	5400	4325	47.8	4422	50.0	4.2
Average Energy Saving				**53.8**	-	**57.3**	**7.8**

management are performed at compile time and hence the genetic algorithm run-time can be tolerated.

We assume all PEs are homogeneous and tasks have similar worst case execution times on each PE. The PEs supports DVS with four different voltages and their corresponding clock frequencies as below: ((1.75V,1000MHz), (1.40V, 800MHz), (1.20V, 600MHz) and (1.00V, 466MHz)).

The experimental results for RG and TG sets are given in Table 1. The last column labelled $\%improv$ shows the percent improvement (in energy reduction) that the integrated CASPER has vs. the non-integrated approach of HGLS + PDP-SPM. RG graphs are mapped to 4- and 6-PE architectures (depending of the graph size) and TG graphs are mapped to a 6-PE system, which is a reasonable scale for a power/energy-sensitive embedded multiprocessor system. As expected, the makespan-driven HGLS (usually) finds better makespans than CASPER (the two columns labelled by μ in Table 1). HGLS's achieved energy consumption, however, are consistently worse than that of CASPER. Even in those instances where both algorithms find similar makespans (e.g. RG3), CASPER is capable of saving more power. This shows that various task assignment and ordering pairs may generate similar makespans, and a non-integrated framework (where the schedule is used as an input to the power-management algorithm) has no way of distinguishing among such solutions on the basis of their energy saving efficiency. On average, HGLS saves 53.8% energy and CASPER saves 57.8%, with a 7.8% improvement over HGLS.

4.2. Heterogeneous System

First, to evaluate the effectiveness of the integration process, we ran HGLS on the tgff task sets and applied the PV-DVS technique to the results for energy optimization. We then ran CASPER on the same task sets with the same amount of run time. Results of these experiments are reported in Table 2. Columns 3 and 4 show the energy reduction achieved by CASPER (with respect to a task execution at nominal supply voltage) and its %improvement over HGLS + PV-DVS's results respectively. One can see that CASPER outperforms HGLS + PV-DVS in energy efficiency by 8.2%. We mention that we give both algorithms the same run time for a "fair" comparison. However, the PV-DVS algorithm is very time consuming and thus the genetic algorithm in CASPER stops before its stopping condition (S8 in Fig. 3) is reached. We observe that the genetic algorithm has not converged for most of cases, including all the cases with negative results in the last column of Table 2.

Next, we compare CASPER framework against GMA + EE-GLSA algorithm using similar configuration (same allocation and constraints). The results are also shown in Table 2. Our results show that the proposed single loop CASPER framework saves 23.3% more energy over GMA+EE-GLSA that uses two nested optimization loops, even when we restrict its run time as explained above. Potentially, the elimination of one loop may also give us large saving in run time.

In summary, we apply the same power management technique (PV-DVS in this case) in all the three algorithms, their difference in energy efficiency indicates that combining task mapping, ordering, scheduling, and power management in the same loop, rather than separate them, yields better solution.

5. Conclusions

In this paper we presented an integrated approach for task mapping and scheduling onto homogeneous and heterogeneous embedded multiprocessors using a genetic algorithm. We employed a solution representation (for our GA) that encodes both task assignment and ordering into a single chromosome and hence significantly reduces the search space and problem complexity. We employed two

Table 2. Energy saving by CASPER, HGLS and GMA + EE-GLSA for benchmarks of [13].

Task Graph	$\|V\|/\|E\|$	CASPER **%Saving**	%improv vs. GMA	%improv vs. HGLS
tgff1	8/9	82.84	41.64	18.53
tgff2	26/43	71.43	46.02	-0.24
tgff3	40/77	73.27	19.33	-32.71
tgff4	20/33	88.40	32.25	85.30
tgff5	40/77	87.46	72.74	31.44
tgff6	20/26	92.68	59.00	-44.01
tgff7	20/27	34.65	8.28	3.74
tgff8	18/26	74.46	7.32	69.33
tgff9	16/15	43.42	-5.33	-18.40
tgff10	16/21	38.48	19.49	-78.52
tgff11	30/29	17.85	-10.70	-4.22
tgff12	36/50	88.37	40.52	86.05
tgff13	37/36	51.75	-24.43	34.26
tgff14	24/33	22.99	7.12	2.48
tgff15	40/63	84.84	80.34	71.20
tgff16	31/56	22.29	-9.40	-0.59
tgff17	29/56	94.63	90.17	3.37
tgff18	12/15	8.04	-31.42	-28.51
tgff19	14/19	17.05	-56.93	-31.22
tgff20	19/25	83.40	29.59	-28.35
tgff21	70/99	85.80	78.67	55.62
tgff22	100/135	36.27	-21.34	-22.64
tgff23	84/151	86.72	65.08	6.08
tgff24	80/112	75.43	12.01	10.72
tgff25	49/92	51.07	33.49	15.85
Avg. Energy Saving		60.54	23.34	8.2

leading power management techniques (for homogeneous and heterogeneous embedded systems) in the fitness function of our genetic algorithm and integrated framework. We experimentally showed that this integrated framework can save on average about 8% more energy compared to a non-integrated technique using the same power management techniques. Our results showed that a scheduling algorithm (HGLS here) if employed in an integrated framework with a power management algorithm, is capable of improving itself with respect to energy efficiency. More broadly, we also showed that a task assignment and scheduling that generate a better makespan do not necessarily save more power, and hence, integrating task scheduling and slack distribution based power management methods is crucial for fully exploiting the energy-saving potential of an embedded multiprocessor implementation. We also evaluated our synthesis framework and showed that it produces solutions with higher energy efficiency than GMA + EE-GLSA, one of the best known techniques.

6. Acknowledgments

A portion of this research was supported by the Advanced Sensors Collaborative Technology Alliance. The Authors would like to thank Michael D. Rinehart (MIT) for his help in programming the Genetic Algorithm, and Shaoxiong Hua (Synopsys Inc.) for kindly providing his PDP-SPM algorithm.

References

[1] A. Al-Maasarani,"Priority-Based scheduling and evaluation of precedence graphs with communication times," M.Sc. Thesis, King Fahd University of Petroleum and Minerals, Saudi Arabia, 1993.

[2] M. A. Al-Mouhamed, "Lower bound on the number of processors and time for scheduling precedence graphs with communication costs," *IEEE Trans. Software Engineering*, Vol. 16, no. 12, pp. 1390-1401, 1990.

[3] S. Hua and G. Qu, "Power Minimization Techniques on Distributed Real-Time Systems by Global and Local Slack Management," IEEE/ACM Asia South Pacific Design Automation Conference, January 2005.

[4] R. C. Correa, A. Ferreira and P. Rebreyend, "Scheduling multiprocessor tasks with genetic algorithms," *IEEE Tran. on Parallel and Distributed Systems*, Vol. 0, pp. 825-837, 1999.

[5] R. Dick, D. Rhodes, and W. Wolf, "TGFF: Task graphs for free," *Proc. Int. Workshop Hardware/Software Codesign*, pp. 97-101, March 1998.

[6] F. Gruian and K. Kuchcinski, "LEneS: Task scheduling for low-energy systems using variable supply voltage processors," *Proc. of Asia and South Pacific Design Automation Conference*, pp. 449-455, Jan. 2001.

[7] N. K. Jha, "Low power system scheduling and synthesis," *Proc. of Int. Conf. on Computer Aided Design*, pp. 259-263, 2001.

[8] Y. Kwok and I. Ahmad, "Benchmarking and comparison of the task graph scheduling algorithms," *Journal of Parallel and Distributed Computing*, Vol. 59, no. 3, pp. 381-422, Dec. 1999.

[9] J. Luo and N. K. Jha, "Power-profile driven variable voltage scaling for heterogeneous distributed real-time embedded systems," *Int. Conf. on VLSI Design*, Jan. 2003.

[10] C. L. McCreary, A. A. Khan, J. J. Thompson, and M. E. McArdle, "A comparison of heuristics for scheduling DAGS on multiprocessors," *Proc. of the Int. Parallel Processing Symp.*, p 446-451, 1994.

[11] R. Mishra, N. Rastogi, D. Zhu, D. Mossé, and R. Melhem, "Energy aware scheduling for distributed real-time systems, " *Int. Parallel and Distributed Processing Symp.*, pp. 243-248, April 2003.

[12] M. Schmitz, B. Al-Hashimi, and P. Eles, "Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems," *Design, Automation and Test in Europe Conference*, March 2002.

[13] M. Schmitz, B. Al-Hashimi, and P. Eles, "Iterative Schedule Optimisation for Voltage scalable Distributed Embedded Systems," *ACM Trans. on Embedded Computing Systems*, vol. 3, pp. 182-217, 2004.

[14] D. Sylvester and H. Kaul, "Power-driven challenges in nanometer design," *IEEE Design and Test of Computers*, pp. 12-21, Nov. 2001.

[15] M. -Y. Wu and D. D. Gajski, "Hypertool: A programming aid for message-passing systems," *IEEE Trans. on Parallel and Distributed Systems*, 1(7), pp. 330-343, July 1990.